TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Gestió remota de serveis de mòbils mitjançant GCM

TITULACIÓ: Enginyeria Tècnica de Telecomunicacions, especialitat Sistemes de Telecomunicació

AUTOR: Marc Sardà Duran

DIRECTOR: Dolors Royo Vallés

DATA:

Títol: Gestió remota de serveis de mòbils mitjançant GCM

Autor: Marc Sardà Duran

Director: Dolors Royo Vallés

Data:

Resum

Aquest projecte tracta sobre el desenvolupament d'un sistema per entendre el servei que ofereix Google, el Google Cloud Messaging per rebre notificacions emergents en dispositius mòbils.

El sistema està format per una aplicació client, desenvolupada amb l'IDE Eclipse i SDK d'Android, i un servidor programat en C# i Visual Studio 2010.
El client serà capaç de registrar-se al servei GCM per tal de rebre notificacions del servidor GCM. El servidor implementat, serà capaç d'enviar missatges al servidor GCM, perquè aquet els reenviï cap el client.

Per provar el sistema desenvolupat, l'aplicació client haurà de ser capaç de connectar i desconnectar el servei Bluetooth del dispositiu, quan rebi un missatge del servidor GCM.

Title: Remote management of mobile services using GCM

Author: Marc Sardà Duran

Director: Dolors Royo Vallés

Date:

Overview

This project is developing a system to understand the service offered by Google, Google Cloud Messaging to receive emerging notifications to mobile devices.

The system consists of a client application, developed with Eclipse IDE and the Android SDK and server programmed in C #, using Visual Studio 2010. The customer will be able to register to GCM service, and receive notifications of GCM server. The implemented server, will be able to send messages to GCM server, and GCM server resend this message towards the client.

To test developed system, the client application will be able to connect and disconnect device Bluetooth service, when receives a message from GCM server.

ÍNDEX

ACRÒNIMS I ABREVIACIONS

GCM	Google Cloud Messaging
SDK	Software Development Kit
IDE	Integrated Development Environment
ADT	Android Development Tools
API	Application Programming Interface

CAPÍTOL 1. INTRODUCCIÓ

1.1 Objectius

L'objectiu d'aquest projecte es conèixer el funcionament del servei que ofereix Google per rebre notificacions emergents en dispositius mòbils, anomenat *Google Cloud Messaging*, *GCM*.

Es veurà la manera d'activar el servei *GCM* gratuïtament, a partir d'un compte de Google i utilitzar-ho en un dispositiu amb sistema operatiu Android.

Es dissenyarà i implementarà un sistema client - servidor, connectats entre si mitjançant un *socket*, on el client es donarà d'alta del servei *GCM*. Seguidament enviarà, a través del *socket*, informació al servidor i aquest ho reenviarà al servidor *GCM*. Per finalitzar, el client haurà de rebre del servidor *GCM* una notificació.

1.2 Estructura de la memòria

El document està dividit en 6 apartats. A continuació es resumeixen cada un d'ells.

Aquest primer capítol es descriuen objectius que s'han proposat per realitzar el treball. Consisteix en una petita introducció per situar-nos.

El segon capítol es fa una pinzellada de la història d'Android, Es escriu el funcionament de sistema GCM i com donar-se d'alta al servei. També es fa un repàs de l'entorn de programació utilitzat.

El tercer capítol es pot veure el disseny del sistema. El disseny i implementació tant de l'aplicació client com de l'aplicació servidor.

En el quart capítol es detallen els resultats i problemes obtinguts, a partir del plantejament inicial de cada una de les parts del sistema.

En el cinquè capítol estan exposades les conclusions finals envers el treball, així com les millores i futures ampliacions del sistema. També es detallen les fonts d'informació utilitzades.

Per últim s'afegeixen uns annexes, que permeten detallar mes algun concepte.

CAPÍTOL 2. CONCEPTES BÀSICS

2.1 Android

Android és un sistema operatiu basat en Linux, dissenyat principalment per dispositius mòbils amb pantalla tàctil, com telèfons intel·ligents i *tablets*.
Va ser desenvolupada principalment per *Android Inc*. y posteriorment comprada per Google l'any 2005. Android es va donar a conèixer l'any 2007 juntament amb la fundació de la *Open Handset Alliance*, un consorci d'empreses de hardware, software i telecomunicacions dedicades a la promoció d'estàndards de codi lliure.

El fet de ser de codi font lliure, fa que el software pugui ser modificat i distribuït lliurement per fabricants de dispositius i contar amb una amplia comunitat de desenvolupadors, que contínuament escriuen aplicacions que amplien la funcionalitat dels dispositius. Aquests factors han permès a Android convertir-se en una de les plataformes mes utilitzades pels telèfons intel·ligents.

Les aplicacions es desenvolupen habitualment en el llenguatge Java amb Android *Software Development Kit* (*Android SDK*). Hi ha altres eines de desenvolupament, incloent un Kit de Desenvolupament per aplicacions o extensions en C , C++ o altres llenguatges de programació.

2.2 Google Cloud Messaging

2.2.1 Que és?

És un servei gratuït implementat per Google, que permet enviar missatges instantanis des d'un servidor cap a un dispositiu mòbil (*Downstream Messaging*) i a l'inversa, des d'un dispositiu mòbil cap a un servidor (*Upstream Messaging*). Aquest servei permet crear aplicacions que requereixin enviar alertes a temps real o aplicacions de xat i missatgeria instantània, com el mateix whatsapp.

Figura 1.1 Logotip de Google Cloud Messaging

En alguns cassos, les aplicacions mòbils necessiten tenir capacitat per notificar a l'usuari determinats esdeveniments que succeeixen fora del dispositiu, normalment en una aplicació web o un servei *online* allotjat en un servidor extern. Per exemple, podríem pensar en les notificacions que mostra el nostre mòbil quant es rep un nou correu electrònic del servidor de correu habitual.

Per aconseguir-ho, podrien sorgir diverses solucions, per exemple mantenir oberta una connexió permanent amb el servidor, de manera que aquest pogués comunicar immediatament qualsevol esdeveniment a la nostra aplicació. Aquesta tècnica, encara que sigui viable i totalment efectiva, requereix de molts recursos oberts constantment al nostre dispositiu, augmentant així el consum de la CPU, memòria i ample de banda necessaris per executar l'aplicació.
Una altre solució seria fer que la nostra aplicació preguntés constantment al servidor si existeix alguna novetat que notificar a l'usuari. Aquesta tècnica necessita molts menys recursos que la opció anterior, però te un inconvenient que pot ser important segons l'aplicació que afecti: qualsevol esdeveniment que es produeixi al servidor no es notificarà a l'usuari fins la pròxima consulta que faci l'aplicació al servidor. Això podria ser molt temps després.

Per solucionar aquest problema, Google va introduir a Android, a partir de la versió 2.2 *Froyo*, la possibilitat d'implementar notificacions de tipus *push*, el que significa que és el servidor el que inicia el procés de notificació. El pot iniciar al mateix moment que s'està produint l'esdeveniment. El client es limita a esperar els missatges sense necessitat d'estar constantment consultant al servidor per veure si existeixen novetats, i sense mantenir una connexió permanentment oberta.

Figura 1.2 Logotip de Android 2.2 Froyo

L'arquitectura de Google ho aconsegueix introduint un nou element al procés, un servidor de missatgeria *push* o *cloud to device* (missatges del núvol al dispositiu). L'anomenarem servidor GCM. Aquest servidor GCM estaria situat entre el servidor o pàgina web i l'aplicació mòbil. Aquest servidor GCM, s'encarregaria de rebre les notificacions enviades des del servidor o pàgina web i fer-les arribar a les aplicacions mòbils dels corresponents dispositius.

El servidor GCM ha de conèixer de l'existència tan de l'aplicació web com de l'aplicació mòbil, hi ho aconsegueix mitjançant un protocol ben definit de registres i autoritzacions entre ells.

2.2.2 Alta de servei GCM

El procés d'alta d'aquest servei és totalment gratuït, però requereix d'un procés previ de registre i posterior generació d'una *ApiKey*.

Per fer-ho s'ha d'accedir a la consola d'*APIs* de Google, a la següent direcció *https://code.google.com/apis/console*. Si es la primera vegada que s'hi accedeix, apareixerà una pantalla de benvinguda amb una opció de crear un nou projecte.

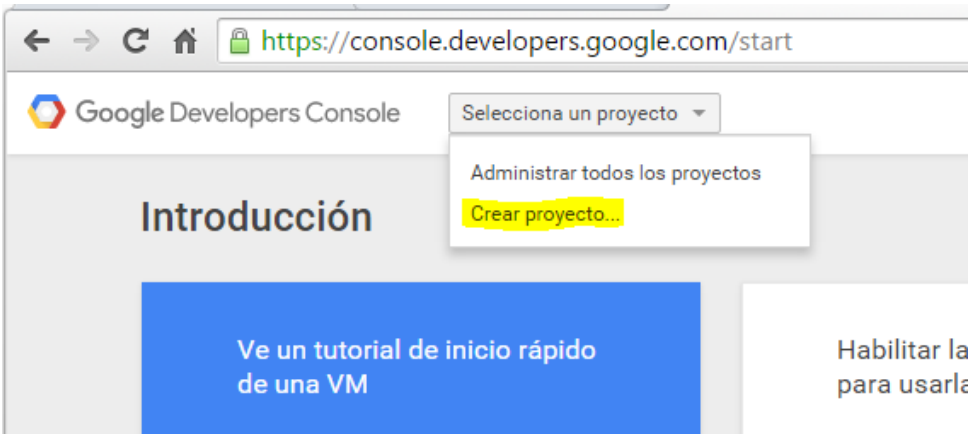


Figura 1.3 Creació de projecte

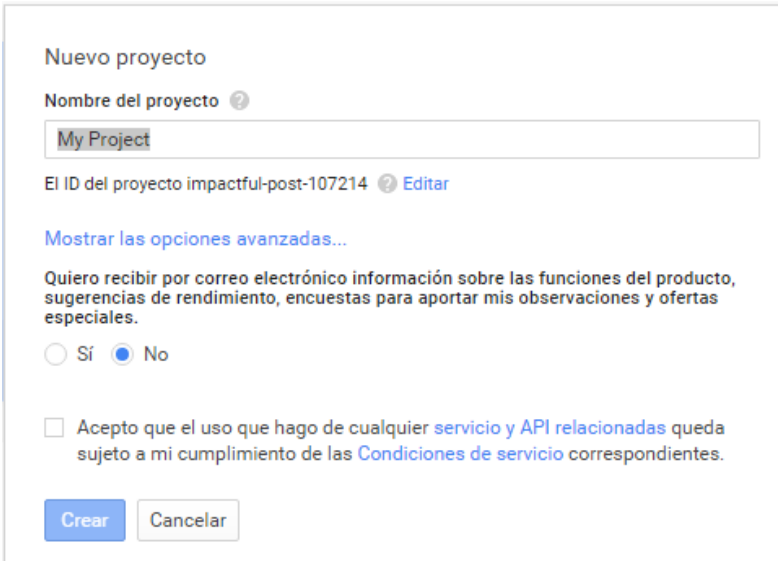


Figura 1.4 Creació d'un nou projecte

Un cop creat el projecte, el navegador ens dirigeix a una direcció similar a aquesta:

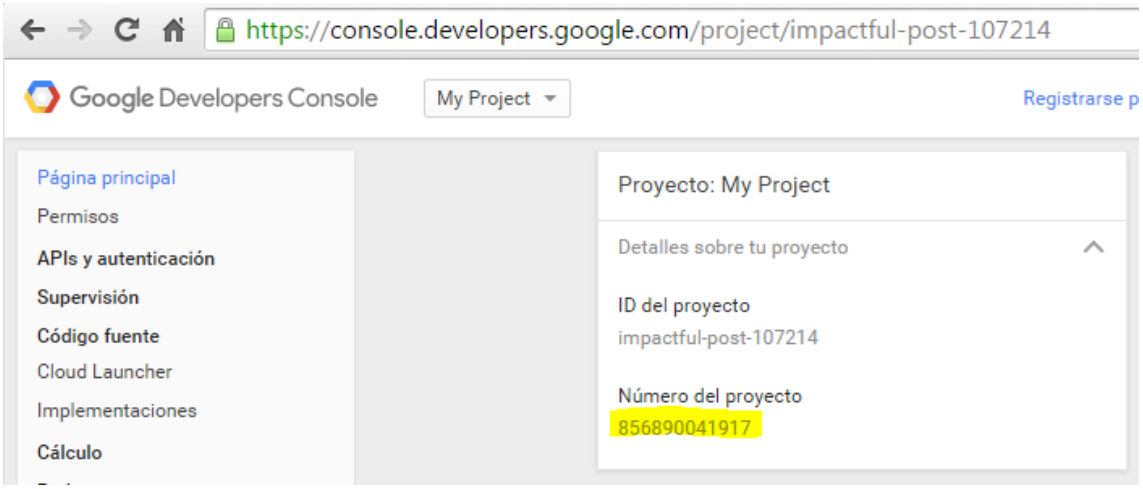


Figura 1.5 Pagina web del projecte

El número que apareix a l'etiqueta Número de Projecte l'anomenarem *Sender ID* i serà l'identificador únic de l'aplicació emissora dels missatges.

Un cop creat el nou projecte, s'ha d'activar el servei GCM. Per fer-ho s'ha d'accedir al menú *APIs i autenticació*, i entrar a l'apartat *APIs,* seguit de *APIs para mòbils.*

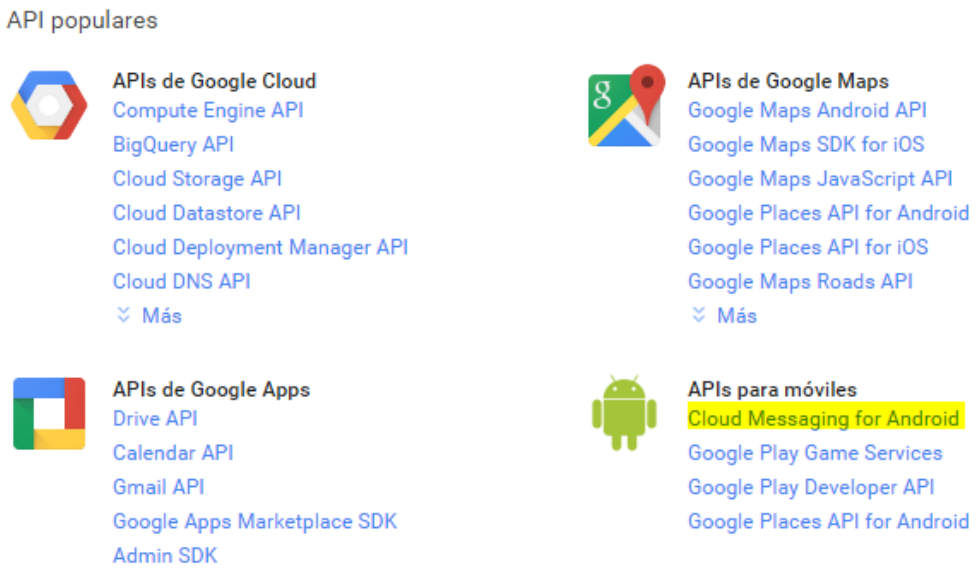


Figura 1.6 API's per a mòbils

En aquesta pantalla, habilitem la API el servei de *Google Cloud Messaging for Android.*

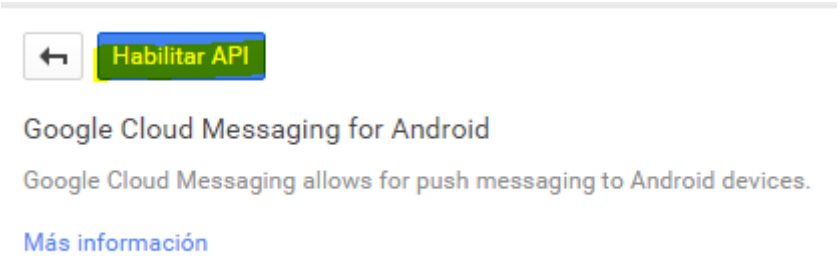


Figura 1.7 Activació del Servei GCM

Un cop realitzades aquestes tasques, ja quedaria el servei GCM activat i associat a un projecte. És important conservar el valor de *Sender Id* i del *Apikey,* seran necessaris en els següents punts del projecte.

2.3 Entorn de desenvolupament

2.3.1 Entorn Android

Per començar a desenvolupar aplicacions per a Android des d'un PC, és necessari instal·lar una sèrie de components que ens donaran el suport necessari per realitzar aquesta tasca.
Concretament, haurem d'instal·lar un entorn de desenvolupament amb *Eclipse*, *l'SDK* d'Android i descarregar el plugin d'Android per *Eclipse*. A continuació es descriu breument cada un d'aquests components:

Eclipse: Es tracta d'un entorn de desenvolupament integrat (*IDE*, *Integrated Delevopment Environment*),que presenta una interfície gràfica dissenyada per a facilitar la gestió i el desenvolupament.

SDK: Les seves sigles signifiquen Kit de Desenvolupament de Software. Es tracta d'un conjunt d'eines que permeten al programador crear aplicacions per a un sistema concret. En aquest cas utilitzarem l'SDK de Java, que és qui dona suport a Android.

Plugin d'Android: *L'ADT* (*Android Development Tools*) és un plugin per a l'IDE Eclipse que està dissenyat per a proporcionar un entorn de desenvolupament integrat amb el que construir aplicacions d'Android.

.

CAPÍTOL 3. DISSENY DEL SISTEMA

Per provar el sistema GCM, és a dir, el servidor de missatgeria o notificacions *push* que ofereix Google (Servidor GCM), s'ha dissenyat una aplicació client (executada en un dispositiu mòbil), i una aplicació servidor (executada en un ordinador portàtil).

L'aplicació client, s'executa des d'un dispositiu mòbil amb tecnologia Android. La tasques fonamentals del client és donar-se d'alta del servei al servidor GCM (per tal de poder rebre notificacions d'aquest) i comunicar-se (mitjançant un *socket*) amb el servidor implementat.

El servidor, que s'executa en un PC, espera la connexió de l'aplicació client (mitjançant un *socket*). Un cop estableixen connexió, el client envia una petició i el servidor ho reenvia cap al servidor GCM. La comunicació entre l'aplicació client i l'aplicació servidor és unidireccional, de client a servidor.

Per acabar d'implementar el sistema, la finalitat de l'aplicació serà activar el servei de Bluetooth del dispositiu. El procés a seguir es el següent:

- El client envia un missatge al servidor.
- El servidor ho fa cap al servidor GCM.
- El servidor GCM envia una notificació al client.
- El client activa el Bluetooth del dispositiu.

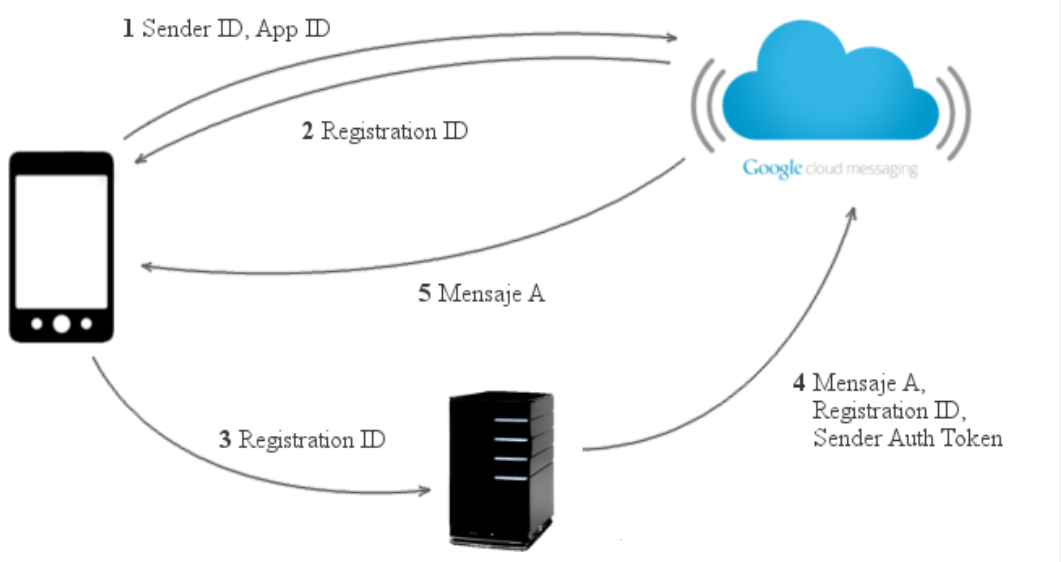


Figura 3.1 Esquema del sistema muntat

3.1 Servidor

3.1.1 Estructura

Com s'ha comentat, el servidor ha de poder processar missatges rebuts del client, interpretar-los i enviar missatges cap al servidor GCM.

Per tant, les funcions del servidor es poden resumir en els següents 3 punts:

- Rebre missatges de l'aplicació client.
- Processar els missatges rebuts.
- Enviar missatges al servidor *GCM*.

Per rebre missatges del client, primer s'ha de connectar el servidor amb el client. Aquesta acció es durà a terme mitjançant un *socket*.
El servidor obre un *socket*, i es queda a l'espera de rebre una petició de connexió per part d'algun client. El *socket* està obert per rebre connexions per el port 5213 (escollit a l'atzar) i des de qualsevol direcció *IP.*

Un cop s'estableix la connexió client-servidor, el servidor rebrà un missatge del client, on apareixerà el *Registration_Id* del client i la tasca que vol realitzar.

El següent pas, és enviar un missatge al servidor GCM. Per poder enviar un missatge al servidor GCM, dirigit a un client en concret, necessitem tres dades essencials:

- *ApiKey*, associada al projecte, coneguda des de l'inici.
- *Registration_Id*, associat al client, rebuda en el pas anterior.
- Missatge a enviar, en el nostre cas serà la tasca a fer.

Enviant aquest missatge al servidor GCM, acabarien les tasques del servidor.

3.1.2 Implementació

Com hem comentat anteriorment, per implementar el servidor s'ha utilitzat el programa Visual Studio 2010.

Per fer-ho, obrim el programa i creem un nou projecte en Visual C# del tipus *Console Application*. L'anomenarem server.

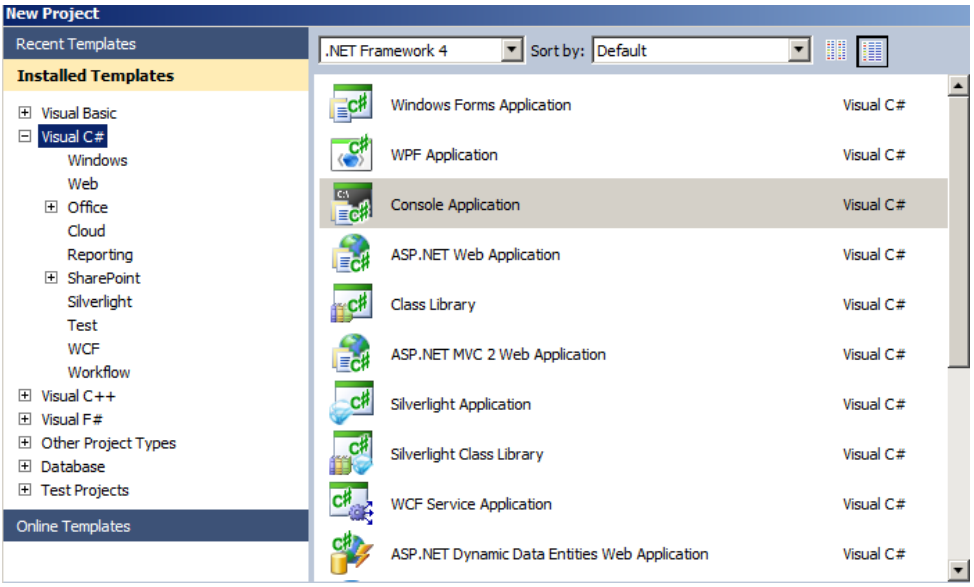


Figura 3.2 Menú de Visual Studio 2010

Per escriure el codi del servidor utilitzarem l'arxiu *Program.cs*. Aquest arxiu apareix automàticament quan es crea el projecte. Modificarem el nom pel de *server.cs.*

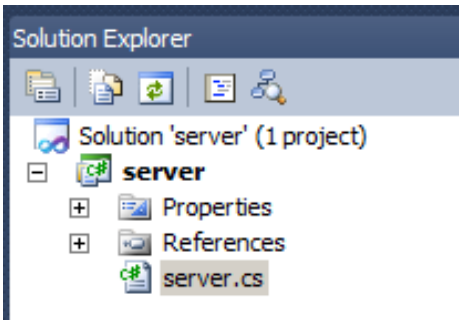


Figura 3.3 Estructura projecte

Per poder utilitzar els *sockets* és necessari utilitzar unes classes especifiques, això ho farem incloent les següents sentències a l'arxiu *server.cs*:

```
using System.Net.Sockets;
using System.Net;
```

Ara ja podem començar inicialitzar el *socket*. Ho farem utilitzant una funció:

```
public void Conectar_socket()
```

Seguint els passos descrits a l'estructura, crearem el socket i definirem les direccions IP que el servidor ha d'escoltar. En aquest cas les escoltarem totes utilitzant *IPAddress.Any*, i utilitzarem el port 5213 escollit a l'atzar.

```
Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);

      IPEndPoint Ep = new IPEndPoint(IPAddress.Any, 5213);
```

Crearem un buffer per rebre els missatges del client:

```
byte[] buffer = new byte[1024];
```

Posem el socket a escoltar peticions. El servidor queda en repòs fins que algun client es connecta.

```
do{
      socket.Listen(100); // escoltem clients
      Console.WriteLine("Esperant accept");

      Socket handler = socket.Accept(); // socket inicialitzat
      Console.WriteLine("Esperant clients");

      count = handler.Receive(bytes); / / Rebem dades clients
```

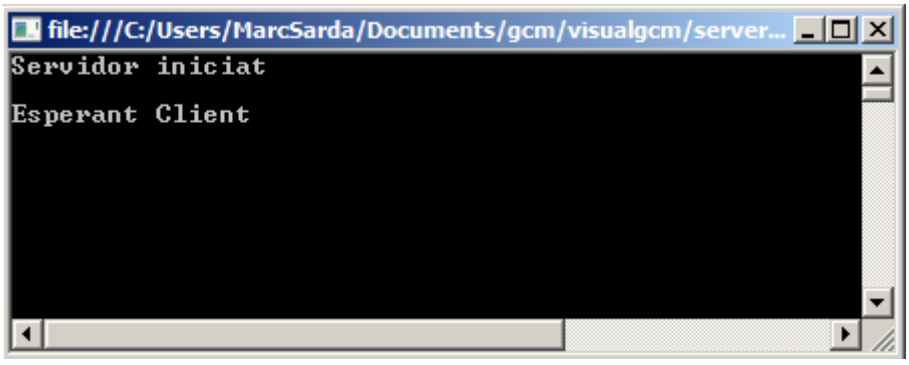


Figura 3.4 Servidor esperant clients

Un cop tenim el client connectat, passem a processar els missatges que rebem. La primera part del missatge serà el *Registration_Id*, i l'últim caràcter serà el caràcter de control. Separarem les dues parts del missatge de la següent manera:

```
// De-Codifiquem la cadena
data = System.Text.Encoding.ASCII.GetString(bytes, 0, count);
```

```
// Control d'errors
if (data.Equals(""))
	Console.WriteLine("El client no està registrat a GCM");

else
{
	// Guardem el digit de control a tasca
	tasca = data[data.Length - 1].ToString();

	// eliminamos caracter que em tret
	data = data.Substring(0, data.Length - 1);
}
```

Es fa un control d'errors, per si el client no està donat d'alta al servei GCM. Si el missatge enviat és una cadena buida, el programa tindria problemes en la següent comanda:

```
	tasca = data[data.Length - 1].ToString();
```

Un cop tenim les dades separades, entre Reg Id i el caràcter de control, enviem el missatge al dispositiu amb aquetes dades i tanquem el socket. S'utilitzaran les següents comandes:

```
CreateGcmMessage(data, tasca);
handler.Close(); //el tanquem
```

La funció *CreateGcmMessge*, el que fa és preparar el missatge que s'enviarà al client a través del servidor GCM. Aquesta funció solament necessita el missatge que s'ha d'enviar i el *Registration_Id* del client. Aquests dos valors els passarem com paràmetres de referencia. La funció és la següent:

```
private void CreateGcmMessage(string deviceId, string mens) {

	string postData = "{ \"registration_ids\": [ ";
	string tickerText = "Activar";
	string contentTitle = "Prova";

	// es monta el missatge a enviar
	postData = postData + "\"" + deviceId + "\" ], " +
	"\"data\": {\"tickerText\":\"" + tickerText + "\", " +
	"\"contentTitle\":\"" + contentTitle + "\", " +
	"\"message\": \"" + mens + "\"}}";

	Console.WriteLine(postData);

	// cridem la sgüent funció per enviar el missatge
	SendGcmMessage(apiKey, postData, "application/json");
}
```

Amb el missatge preparat per enviar, es crida la funció *SendGcmMessage*. El que es necessita per enviar el missatge al servidor GCM, arribat aquest punt, és el *apiKey* (resultant del registre que previ al servei GCM), el *Registration_Id* que identifica el client i el missatge a enviar. El codi quedaria de la següent manera:

```
public static void SendGcmMessage(string apiKey, string postData, string
postDataContentType)
{
	byte[] byteArray = Encoding.UTF8.GetBytes(postData);
	HttpWebRequest Request =
(HttpWebRequest)WebRequest.Create("https://android.googleapis.com/gcm/send");

	Request.Method = "POST";
	Request.KeepAlive = false;
	Request.ContentType = postDataContentType;
	Request.Headers.Add(string.Format("Authorization: key={0}", apiKey));
	Request.ContentLength = byteArray.Length;

	Stream dataStream = Request.GetRequestStream();
	dataStream.Write(byteArray, 0, byteArray.Length);
	dataStream.Close();

	try
	{
		WebResponse Response = Request.GetResponse();
		HttpStatusCode ResponseCode =
((HttpWebResponse)Response).StatusCode;

	if (ResponseCode.Equals(HttpStatusCode.Unauthorized) ||
ResponseCode.Equals(HttpStatusCode.Forbidden))
	{
		Console.WriteLine("Unauthorized.");
	}
	else if (!ResponseCode.Equals(HttpStatusCode.OK))
	{
		Console.WriteLine("Response isn't OK.");
	 }
	 else if (ResponseCode.Equals(HttpStatusCode.OK))
	{
		Console.WriteLine("Response is OK.");
	}

	StreamReader Reader = new StreamReader(Response.GetResponseStream());
	string responseLine = Reader.ReadToEnd();

	Reader.Close();

	Console.WriteLine(responseLine);
	}
	catch (Exception e)
	{
	Console.WriteLine(e.ToString());
	}
}
```

Per tant, l'estructura bàsica del servidor seria la següent.

```
public Program()
{
	do {
		Conectar_socket();
		Console.WriteLine("Desconectar el servidor? SI / NO");
		cont = Console.ReadLine();

	} while (!cont.Equals("SI"));
}
```

On des de la funció a *Conectar_socket()* es desenvolupa tot el programa ja que es criden a les funcions *CreateGsmMessage()* i aquesta crida a *SendGcmMessage().* Un cop es tenca el socket, ens apareix per consola el missatge per si es vol continuar o es vol tancar el servidor.

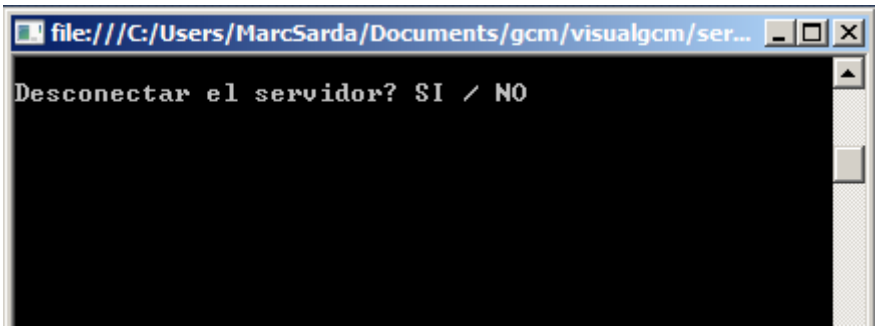


Figura 3.5 Pantalla final del servidor

3.2 Client

3.2.1 Estructura

L'aplicació client ha de ser capaç d'enviar missatges a l'aplicació servidor (detallada al punt anterior) i de rebre missatges del servidor GCM. Per fer-ho, es pot dividir l'estructura de l'aplicació de la següent manera:

- Registrar-se al servidor GCM i obtenir el *"Registration_Id".*
- Connectar-se amb l'aplicació servidor.
- Rebre i processar els missatges provinents del servidor de GCM.

El primer pas és el de registrar-se al servei GCM com un usuari que pot rebre missatges del servidor GCM. Ho farem prement el botó de *Reg GCM*. Al fer-ho obtindrem el *Registration_Id* que el servei GCM ha associat al terminal. Per registrar-se, ens farà falta el *Sender Id*, aquest valor s'obté juntament amb l'*ApiKey* durant el registre del projecte als servidors de Google (explicat a l'apartat 2.1.2). Per confirmar que el registre s'ha fet correctament, rebrem una notificació a la barra d'estat on podrem veure el "Registre OK".
També tenim l'opció de des-registrar-nos del servei GCM premem el botó *Des GCM*. Al clicar el botó, rebrem una notificació confirmant que ens hem des-registrat correctament del servei GCM.

Un cop registrats al servei GCM, és l'hora de comunicar-nos amb l'aplicació servidor. Com s'ha comentat en l'apartat anterior, aquesta connexió es realitzarà mitjançant un socket. L'aplicació ja es coneixedora de la direcció IP del servidor i del port que està habilitat. Prement el botó de *Connectar Servidor* realitzarem una prova de connexió del *socket*.
Després d'establir connexió amb el servidor, s'enviarà un missatge amb el *Registration_Id* i un numero de control que codifica cada una de les accions que es poden dur a terme, segons el botó que es premi.
Tenim dos botons per prémer, el de *Bluetooth* i el *Wi-fi*.

Seguidament, si tot és correcte, hem de rebre un missatge del servidor GCM, el qual processarem i s'activarà el *Bluetooth* del dispositiu.

Per ordenar i aclarir els botons que ens trobarem al obrir l'aplicació, aquí tenim una taula resum;

Botó	**Funció**
Reg GCM	Enviar la sol·licitud de registre al servei GCM.
NO GCM	Anul·lar el registre al servei GCM.
Connectar Servidor	Prova de connexió mitjançant un socket amb el servidor

Bluetooth	Activar o desactivar el bluetooth del dispositiu. Es rebrà una notificació si s'activa el bluetooth "Bluetootht ON" o si es desactiva "Bluetooth OFF"
Wi-fi	El rebrà una notificació amb el text "Wi-fi ON" o "Wi-fi OFF". Aquest botó servirà per una futura ampliació o millora de l'aplicació.
ID	Premem aquest botó podrem veure el *Registration_Id,* si estem registrats.

Figura 3.6 Taula explicativa del botons de l'aplicació

Quant l'aplicació rep un missatge del servidor GCM, apareix una alerta a la barra d'estat, aquesta alerta no és més que un missatge explicatiu de les modificacions que va tenint el dispositiu.

3.2.2 Implementació

Utilitzarem programa Eclipse i Android SDK per programar el client. Crearem un nou *Android Application Porject*.

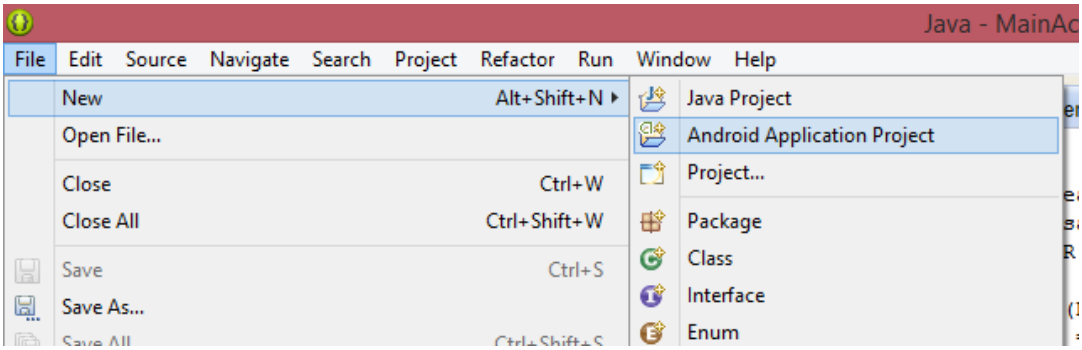


Figura 3.7 Creem projecte nou

L'aspecte final de l'aplicació és la següent serà la següent:

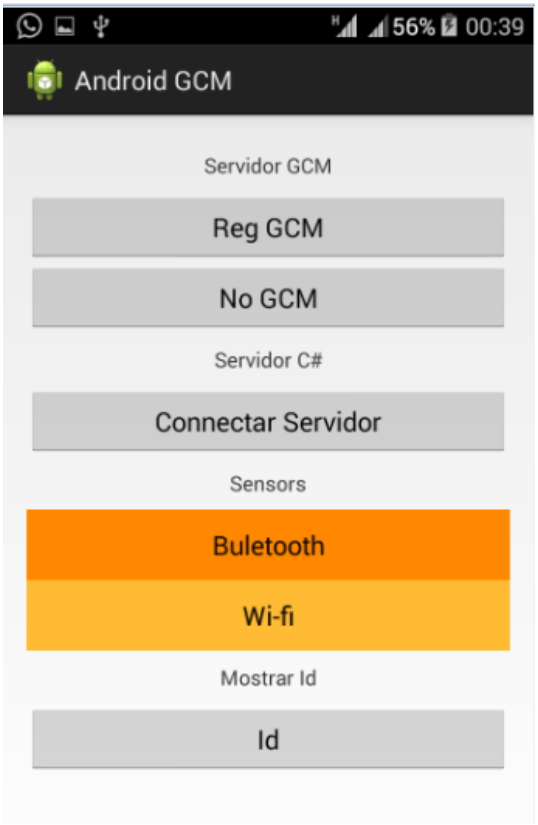


Figura 3.8 Aspecte de l'Aplicació.

A partir d'aquí, anem explicant per parts cada un dels apartats. El primer és donar d'alta el servei GCM. Mitjançant el botó *Reg GCM*, el primer botó que ens trobem, fem la petició de registre al servei GCM.

```
btnRegistrar.setOnClickListener(new OnClickListener() {

	@Override
	public void onClick(View v) {

	// Comprobem que no estiguem registrats
	final String regId = GCMRegistrar.getRegistrationId
	(MainActivity.this);

	if (regId.equals("")) {

		GCMRegistrar.register(MainActivity.this, "1056556668497");
		btnRegistrar.setText("Registrat");
		btnDesRegistrar.setText("No GCM

	} else {

		btnRegistrar.setText("Registrat");
		btnDesRegistrar.setText("No GCM");
	}
	}
}
```

Per fer-ho es crida a un mètode de la classe *GCMRegistrar*, ja definida a la llibreria *gcm.jar*. Passem per referència el *Sender_Id*. Recordem que el *Sender_Id* és valor obtingut junt amb l'*Apikey* al registrar el projecte al servei GCM. Tant la resposta a la petició de registre com la posterior recepció de missatges es fan en forma de *intents*.

A l'arxiu *AndroidManifest.xml*, ubicat a l'arrel de la carpeta del projecte, hem declarat un *boradcast receiver* anomenat *GCMBroacastReceiver*, aquest serà l'encarregat d'esperar i capturar els intents quan es rebin.

Posteriorment es llançarà el servei *GCMIntentService*. La implementació de *IntentService* si que és responsabilitat nostre. Per fer-ho però, la llibreria GCM ens proporciona certa ajuda mitjançant una classe base de nom *GCMBaseIntentService*, el qual te definides una sèrie de mètodes:

- *onRegistered (context, regId):* es crida al rebre una resposta correcta a la petició de registre, a part inclou com a paràmetre el *Registration_Id* assignat al nostre client.

- *onUnregistered (context, regId):* exactament idèntic a l'anterior però aplicat a una petició de des-registrar-se

- *onError (context, errorId):* es crida al rebre una resposta d'un error a la petició de registre o des-registre. El codi concret de l'error es rep com un paràmetre de la funció.

- *onMessage (contect, intent):* es crida cada cop que es rep un missatge nou missatge des de el servidor GCM, el contingut del missatge es rep en forma d'intent.

Al donar-nos d'alta del servei, l'aspecte visual de l'aplicació canviarà lleugerament. El botó *Reg* GCM canviarà per *Registrat* i rebrem una notificació indicant que estem el registre ha estat correcte.

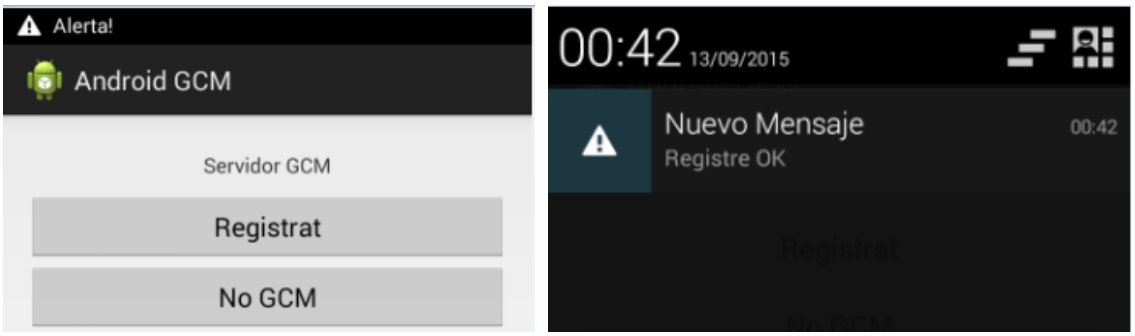


Figura 3.9 Botó Registrar-se al servei GCM

De la mateixa manera, quant premem el botó de *No GCM*, ens des-registrarem del servei cridant a la funció *unregister().* En aquest cas, el botó *No GCM* passarà a *Desregistrat*.

```
btnDesRegistrar.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {

      //Si estem registrats Ens des-registrem de GCM
      final String regId = GCMRegistrar.getRegistrationId
      (MainActivity.this);

      if (!regId.equals("")) {
            GCMRegistrar.unregister(MainActivity.this);
            btnRegistrar.setText("Reg GCM");
            btnDesRegistrar.setText("Desregistrat");
      } else {
            btnRegistrar.setText("Reg GCM"); // prova i error
            btnDesRegistrar.setText("Desregistrat");
      }
      }
});
```

També tindrem una notificació conforme el procés s'ha executat correcte.

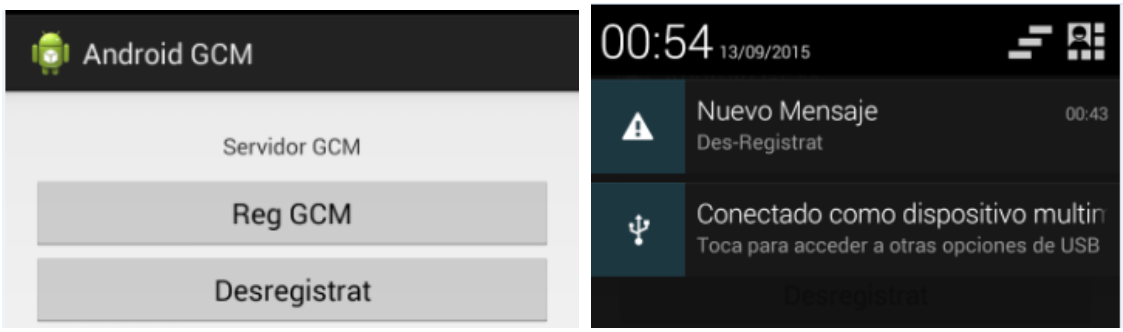
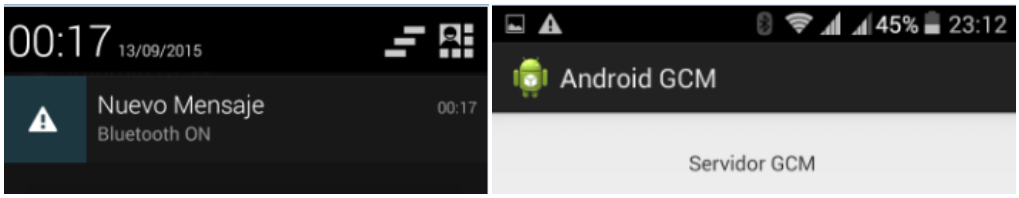


Figura 3.10 Desregistrat de servei GCM

Quant premem el botó Bluetooth, s'activarà o desactivarà el servei de bluetooth del dispositiu, segons estigui encès o aturat. Rebrem una notificació per indicar l'estat del bluetooth, un cop premut el botó.



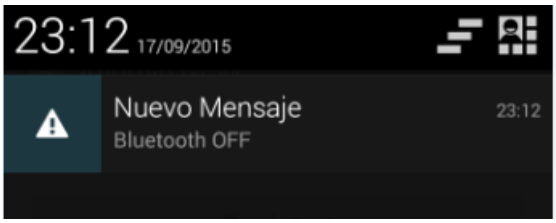


Figura 3.11 Activació/desactivació el servei bluetooth.

Es deixarà preparat el botó de Wi-fi per una futura ampliació i/o modificació de l'aplicació. En tot cas, si premem el botó, hauríem de rebre una notificació on s'indicarà que s'estan realitzant proves amb Wifi del dispositiu.

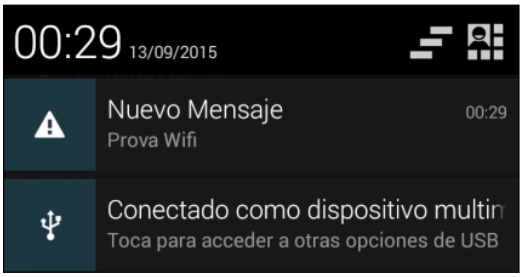


Figura 3.12 Proves del servei Wi-fi

Per mostrar les notificacions a la barra d'estat, utilitzarem la funció *mostrarNotificación.*

```
private void mostrarNotificacion(Context context, String msg)
{
      //Obtenim una referencia al servei de notificació
      String ns = Context.NOTIFICATION_SERVICE;
      NotificationManager notManager =     (NotificationManager)
context.getSystemService(ns);

      //Configurem la notificació
      int icono = android.R.drawable.stat_sys_warning;
      CharSequence textoEstado = "Alerta!";
      long hora = System.currentTimeMillis();

      Notification notif = new Notification(icono, textoEstado, hora);

      //Configurem el Intent
      Context contexto = context.getApplicationContext();
      CharSequence titulo = "Nuevo Mensaje";
      CharSequence descripcion = msg;

      Intent notIntent = new Intent(contexto,
      GCMIntentService.class);

      PendingIntent contIntent = PendingIntent.getActivity(
```

```
        contexto, 0, notIntent, 0);

        notif.setLatestEventInfo(
        contexto, titulo, descripcion, contIntent);

        //Quan premem la notificació desapareix
        notif.flags |= Notification.FLAG_AUTO_CANCEL;

        //Enviar notificació
        notManager.notify(1, notif);
}
```

Hi ha un botó a la part inferior de la pantalla, el qual si el premem ens apareixerà el *Registration_Id*, si estem registrats. És útil per realitzar proves amb el servidor, en una versió final aquest botó desapareixeria.

Figura 3.13 Mostrem el *Registration_Id*

Els següent pas, és el de comunicar-se amb el servidor. Ho farem mitjançant un socket, premen el botó de *Connectar Servidor*. Prèviament ja sabem a quina direcció IP te el servidor i en quin port esta escoltant les peticions. Cal dir que el servidor ha d'estar inicialitzat.

Figura 3.14 Botó per provar el socket

La funció que inicialitza el socket és la següent.

```
// Conectarem amb servidor i mitjançant un socket
btnSocket.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
```

```
            String ip = "192.168.43.91";
            int puerto = 5213;      //enviem missatge al servidor
      try {

            socket = new Socket( ip, puerto);

            if (socket.isConnected())
                  btnSocket.setText("Socket ok");
            else
                  btnSocket.setText("ko");

            writer = new PrintWriter(socket. getOutputStream());
            writer.print(mensaje);
            writer.flush();
            socket.close();

            }

            catch (Exception e) {
                  e.printStackTrace();
                  btnSocket.setText("Socket ko");
            }
      }
});
```

El mateix botó canviarà de nom segons el resultat obtingut. Si hi ha algun error en la inicialització del socket, ens retornarà un error. "Socket ko". En cas contrari, si ha estat possible inicialitzar la connexió el missatge del botó serà favorable "Socket ok".

Aquesta funció és la que s'utilitza cada cop que es vol passar informació al servidor. Cada cop que crida, es crea un socket, s'envia la informació i es torna a tancar.

De la mateixa manera, rebrem una notificació indicant que el socket és correcte:

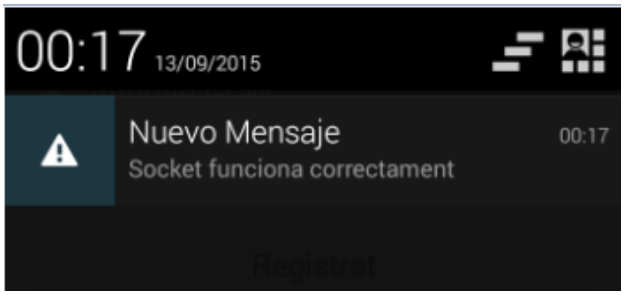


Figura 3.15 Notificació de funcionament de socket.

CAPÍTOL 4. RESULTATS OBTINGUTS

4.1 Introducció

El servidor s'ha programat i executat en un ordinador portàtil Toshiba amb sistema operatiu Windows 7, des del programa Visual Studio 2010, com una aplicació de consola.

El programa client s'ha compilat amb un arxiu .apk (*Application Package File*) i instal·lat en un dispositiu mòbil BQ Aquaris E5, amb versió Android 4.4.4 Kit Kat.

Inicialment per proveir tant el client com al servidor d'accés a internet, s'ha fet mitjançant les dades 3G d'un dispositiu mòbil (BQ Aquaris E5 i Samsung Galaxy S4 Mini).

4.2 Resultats

Per realitzar la connexió, mitjançant un *socket* entre el client i el servidor, és on he trobat el primer problema. El servidor s'executa des d'un ordinador portàtil, el qual, com he comentat anteriorment, accedeix a la xarxa mitjançant les dades 3G d'un dispositiu mòbil.

Inicialment vaig pensar que el mòbil actuaria com un router convencional, i que podria crear una xarxa local per connectar el servidor i el client. El primer dispositiu utilitzat com a router, un mòbil BQ Aquaris E5, no ha estat possible accedir als ports necessaris i implementar el *socket*. Aquest inconvenient es va resoldre utilitzant un Samsung Galaxy S4 Min.

Ara veurem les proves realitzades de cada una de les opcions que té el client en la pantalla de l'aplicació:

4.2.1 Connectar al Servidor

Un cop resolt l'inconvenient de crear la xarxa local, s'ha trobat un problema d'incompatibilitat de la versió del SDK, els sistema operatiu Android i les polítiques internes, la quals cosa feia que la creació del socket retornes un error. S'ha resolt introduint les següents línies de codi:

```
if (android.os.Build.VERSION.SDK_INT > 9) {
      StrictMode.ThreadPolicy policy = new StrictMode.
ThreadPolicy.Builder().permitAll().build();
      StrictMode.setThreadPolicy(policy);
}
```

Comprovem el funcionament premen el botó de “Connectar Servidor”. El resultat és satisfactori. Veiem que els servidor rep la petició de connexió del client, i el client rep la notificació de “Socket funciona correctament”.

Confirmem que la connexió mitjançant un socket entre client i servidor es correcte.

4.2.2 Registre GCM

Si premem el botó “Reg GCM”, veiem que el ens surt una notificació de “Registre ok”. Per acabar de verificar que hem rebut i guardat el codi *Registration_Id* premem el botó *ID,* introduït a l’aplicació per comprovar que el registre al del servei GCM és correcte.

El resultat és l’esperat, premem el botó “ID”, ens apareix el valor de *Registration_Id* com a text del mateix botó.

4.2.3 Des-Registre GCM

Per comprovar si la funció de des-registrar-nos del servei GCM funciona correctament, premem el botó de “Des GCM”. Tot seguit rebem una notificació de “Des-registrat”.

Si premem el botó de “ID”, el text desapareix, la qual cosa indica que no el *Registration_Id* és nul.

4.2.4 Activar Bluetooth

Per comprovar la funció d’activar o desactivar el Bluetooth, partim de la base que el Bluetooth del dispositiu està desactivat.

Si premem el botó de “Bluetooth”, tot seguit rebem una notificació de “Bluetooth ON” i també verifiquem que està activat mirant el ajustos del mòbil.

Si tornem a premem el botó, veiem que rebem el missatge de “Bluetooth OFF” i verifiquem que està desactivat als ajustos del mòbil.

4.2.5 Proves Wifi

El botó de “Wi-fi” s’ha deixat preparat per una futura ampliació o modificació del l’aplicació. En tot cas, s’ha programat per tal que quan es premi el botó, rebem un missatge de “Prova Wifi”.
Verifiquem que rebem el missatge en premem el botó.

4.3 Objectius assolits

L’objectiu plantejat a l’inici del projecte era conèixer el funcionament del servei GCM. Per fer-ho, s’ha dissenyat i provat el sistema explicat en els punts anterior.

En les proves realitzades, hem comprovat el correcte funcionament les tasques associades a cada part del sistema.

Per la part del client, objectius eren:

- Donar-se d’alta al servei GCM
- Connectar-se mitjançant un socket amb el servidor
- Enviar informació al servidor a través del socket
- Rebre missatges del servidor GCM

Per la part del servidor, els objectius eren:

- Inicialitzar un socket i escoltar peticions de clients
- Acceptar la petició de connexió d’un client
- Rebre un missatge del client a través del socket
- Enviar un missatge al servidor GCM

S’ha verificat el correcte funcionament de totes les tasques definides per entendre i provar el sistema GCM, per tant, podem concloure que l’objectiu inicial del projecte s’ha assolit

CAPÍTOL 5. CONCLUSIONS

5.1 Anàlisi

La valoració que faig del projecte és molt satisfactòria. Vaig escollir una temàtica que divergeix força de les assignatures que es veuen en l'especialitat de Sistemes de Telecomunicació. Una temàtica que al cap dels anys i després de fer un repàs de totes les matèries cursades, he vist que m'atrau força i em fa pensar que potser em vaig equivocar a l'hora d'escollir especialització i hagués hagut d'escollir la branca telemàtica.

Pel que fa el projecte en sí, personalment la conclusió final és més que satisfactòria. Ha estat molt enriquidor conèixer des de zero com crear una aplicació per un dispositiu Android, com es pot comportar el dispositiu intercanviant informació amb d'altres dispositius i veure el funcionament del servei que ofereix Google, el Google Cloud Messaging per rebre notificacions emergents al nostre dispositiu.

D'altra banda, a mesura que anava avançant el projecte, me'n he adonat que en programació, quant estem pensant en com escriure una línia de codi per executar una tasca, sempre hi ha una manera mes eficient o òptima per executar aquella tasca. Sempre es pot millorar el codi ja sigui simplificant-lo, ordenant-lo o utilitzant altres maneres per aconseguir arribar al mateix punt. Per aclarir-ho, podem dir que no és el mateix sumar que multiplicar, però el resultat final si és el mateix.

Penso que en aquest projecte es pot aprofundir i millorar en diversos aspectes. El sistema de Google Cloud Messaging té molt potencial i es podrien fer diverses aplicacions utilitzant aquest servei.

5.2 Línies futures

Com he comentat en el punt anterior, hi ha moltes coses a millorar del projecte. Sobre el servidor, una millora important seria poder acceptar mes d'un client, guardant el nom i identificacions de cada client que es connectés. Seria interessant que prengués decisions per ell mateix, és a dir, que no actués solament com un repetidor de missatges i que sense rebre cap missatge d'un client connectat poguessin enviar missatges al servidor GCM.

Seguint amb la línia que pogués prendre decisions, podria desenvolupar-se com una aplicació de Windows en comptes de una aplicació de consola. Fent-ho molt mes visual i intuïtiu per l'usuari.

Pel que fa el client, es pot millorar la interfície d'usuari, mes agradable. S'haurien d'implementar controls d'errors a cada intercanvi d'informació entre dispositiu i servidors.

Un altre millora seria comprovar si sensor Bluetooth del dispositiu està activat al iniciar l'aplicació, ja que en aquesta versió, es dona per fet que el sensor està desactivat.

Ja s'ha deixat implementada una millora per una futura ampliació de l'aplicació, es tracta de la funció d'activar i desactivar el sensor wifi. No s'ha desenvolupat en aquest versió, ja que no és una tasca lligada directament amb el servei de GCM.

5.3 Fonts d'informació

- Descàrrega de Visual Studio
  - https://www.visualstudio.com/downloads/
- Manual para la instal·lació del SDK d'Android
  - http://developer.android.com/sdk/installing.html
- Introducción sobre Android
  - http://open.movilforum.comblog/introduccion-android
- Manual instalación Android
  - http://www.javamovil.info/J2ME/android_1.html
- Libro sobre Android
  - http://andbook.anddev.org/files/andbook.pdf
- Registre del servei GCM
  - https://code.google.com/apis/console
- Descàrrega IDE Eclipse
  - http://www.eclipse.org/downloads/

ANNEXA

I. Teoria de Sockets

Igual que el protocol IP permet la comunicació entre ordinadors connectats a Internet, els ports permeten la comunicació entre les aplicacions que s'executen en els ordinadors (tota aplicació proporciona un o més serveis).

En essència un port és la direcció de 16 bits associada, normalment, a una aplicació o servei. Una interfície d'ordinador de xarxa està dividida en 65536 ports (els primers 1024 solen estar reservats per serveis estàndards i processos del sistema operatiu). La divisió no és física, aquests ports no tenen res a veure amb els ports físics del hardware.

En una arquitectura client-servidor, el programa servidor es manté escoltant les peticions de tots els clients a través dels ports que te actius. La informació o els serveis sol·licitats son enviats pel servidor als clients, en concret als ports que aquests estiguin utilitzant per realitzar les peticions.

En una comunicació a través d'Internet, la direcció IP identifica de manera única l'ordinador al que es dirigeix el missatge i el número de port identifica a quin port o aplicació s'estan enviant les dades.

En la família de protocols TCP/IP, existeixen dos protocols de transport (TCP i UDP) que s'encarreguen d'enviar dades d'un port a un altre per fer possible la comunicació entre aplicacions.

El protocol TCP (Transmission Control Protocol) és un servei de transport orientat a la connexió. Durant l'existència de la comunicació hi ha un enllaç entre l'emissor i el receptor, enllaç que permet l'intercanvi bidireccional d'informació. El protocol inclou sistemes de detecció i correcció d'errors i garanteix que els paquets arriben en ordre al destí.

El protocol UDP (User Datagram Protocol) és un servei de transport sense connexió; el missatge de l'emissor es descompon en datagrames UDP i cada datagrama s'envia al receptor pel camí més oportú en aquell moment. Els datagrames poden arribar en qualsevol ordre o no arribar.

Tant TCP com UDP utilitzen sockets (literalment traduït com a endolls) per comunicar programes entre si en una arquitectura client-servidor. Un socket és un punt terminal o extrem en l'enllaç de comunicació entre dos aplicacions (que normalment s'executen en ordinadors diferents). Les aplicacions es comuniquen mitjançant l'enviament i recepció de missatges mitjançant els sockets. Un socket TCP es podria comparar a un telèfon, anàlogament un socket UDP seria la bústia de correus.

A part de classificar-se en sockets TCP i UDP, els sockets poden pertànyer a algun d'aquests dos grups:

- Sockets actius: Poden enviar i rebre dades a través d'una connexió oberta.

- Socksts passius: Esperen intents de connexió. Quan arriba una connexió entrant, li assignen un socket actiu. No serveixen per enviar o rebre dades.

Els socket son creats pel sistema operatiu i ofereixen una interfície de programació d'aplicacions (API) mitjançant la qual les aplicacions poden enviar missatges a altres programes, ja sigui en local o remotament.
Les operacions dels sockets (enviar, rebre, etc.) s'implementen com a crides al sistema operatiu en tots els sistemes operatius actuals. Dit d'una altre manera; els sockets formen part del nucli del sistema operatiu. En els llenguatges orientats a objectes com el Java o el C#, les classes de sockets s'implementen sobre les funcions que dona l'API del sistema operatiu per a l'ús de sockets.

Per fer possible la comunicació client-servidor, el client ha d'establir sockets i connectar-los a sockets del servidor. Els passos que es segueixen en una comunicació típica amb sockets TCP són aquests:

- Es creen els sockets en el client i el servidor.

- El servidor estableix el port pel qual proporcionarà el servei.

- El servidor es mantén escoltant les possibles peticions dels clients pel port predefinit en el pas anterior.

- Un client connecta amb el servidor.

- El servidor accepta la connexió.

- Es realitza l'intercanvi de dades.

- El client o el servidor, o ambdós, tanquen la connexió.

En primer lloc, el servidor ha d'estar en execució abans de que els clients intentin connectar-se a ell, i ha de mantenir actiu un socket passiu en tot moment que escolti qualsevol possible petició entrant.

En segon lloc, el client ha de crear un socket actiu en el qual s'especifiquin la direcció IP i el número de port de l'aplicació que proporciona el servei desitjat. És mitjançant aquest socket que el client comença la comunicació TCP amb el servidor.

En tercer lloc, l'intent de connexió TCP desencadena el fet que en el servidor es crea un socket actiu dedicat única i exclusivament a aquest client (durant el temps de vida d'aquest socket, no podrà ser assignat a cap altre client).

Finalment s'estableix la connexió TCP entre el socket del client i el socket actiu del servidor. A partir d'aleshores, un i altre poden enviar dades o rebre-les a través de la connexió establerta.

II. Instal·lació de l'entorn de Programació Androd

Descàrrega i configuració d'Eclipse

Si s'ha seleccionat Eclipse com a entorn de desenvolupament, es procedirà a descarregar la darrera versió, la 4.5 (Eclipse Mars). Es pot trobar a:

http://www.eclipse.org/downloads/

Es recomana descarregar la versió Eclipse IDE for Java Developers. També la versió apropiada pel sistema operatiu de que es disposi (Windows / Mac OS / Linux, i 32 o 64 bits).

La instal·lació consisteix simplement en descomprimir el zip descarregat en la ubicació desitjada. Per a executar-ho, s'accedirà al fitxer eclipse.exe dins de la ruta on s'hagi descomprimit l'aplicació, per exemple c:\eclipse\eclipse.exe. Durant la primera execució de l'aplicació, es demanarà quina serà la carpeta on s'han d'emmagatzemar els projectes desenvolupats. S'indicarà la ruta desitjada i es marcarà el check "Use this as the default" per a que no ho torni a demanar.

Descàrrega i configuració d'Android SDK

L'SDK de la plataforma Android es pot descarregar des del lloc web:

http://developer.android.com/sdk/index.html

Una vegada descarregat,executar l'instal·lador estàndard de Windows.

Descàrrega i instal·lació de l'ADT

Es pot descarregar mitjançant les opcions d'actualització d'Eclipse, accedint al menú de l'IDE Eclipse "Help / Install new software..." i indicant la següent URL de descàrrega: https://dl-ssl.google.com/android/eclipse/.

Es seleccionaran els dos paquets disponibles, Developer Tools i NDK Plugins. Una vegada instal·lat el plugin, es tindrà que configurar indicant la ruta en la que s'ha instal·lat l'SDK d'Android. Per això, anirem a la finestra de configuració d'Eclipse (Window / Preferences...), i en la secció d'Android indicarem la ruta en la que s'ha instal·lat. Finalment, es pitjarà el botó OK per a acceptar els canvis introduïts.

Descàrrega i instal·lació dels Platforms Tools

A més de l'SDK d'Android ja comentat, el qual conté les eines bàsiques per a desenvolupar en Android, també s'haurà de descarregar les anomenades Platforms Tools, les quals contenen eines específiques de la darrera versió de la plataforma i una o vàries plataformes (SDK Platforms) d'Android, que no són més que les llibreries necessàries per a desenvolupar sobre cada una de les versions concretes d'Android.

Així, si volem desenvolupar, per exemple, per a Android 2.2, tindrem que descarregar la plataforma corresponent. Com a consell personal podem dir que és sempre millor instal·lar dues plataformes: la corresponent a la darrera versió disponible d'Android i, la corresponent a la mínima versió d'Android que volem que suporti l'aplicació desenvolupar.

Per això, des d'Eclipse es pot accedir al menú Window / Android SDK Manager. En la llista de paquets disponibles seleccionarem les Android SDK Platform-tools, les plataformes Android 6 i Android 2.2 (API 8), i el paquet extra Android Support Library, que és una llibreria que ens permetrà utilitzar en versions antigues d'Android característiques introduïdes per versions més recents.

A l'hora de provar i depurar aplicacions Android, no s'haurà de fer necessàriament sobre un dispositiu físic, si no que es pot configurar un emulador o dispositiu virtual (Android Virtual Device, o AVD) on es poden realitzar aquestes tasques. Es pot accedir a l'AVD Manager (menú Window / AVD Manager) i a la secció Virtual Devices es podrà afegir tants AVD com es necessitin (per exemple, configurats per a diferents versions d'Android o diferents tipus de dispositius).

Novament, s'aconsella configurar dos AVDs, un per a la mínima versió d'Android que es vulgui suportar, i l'altre per a la versió més recent de que es disposi.

Per a configurar l'AVD només s'haurà d'indicar un nom descriptiu, la versió de la plataforma Android que s'utilitzarà i les característiques de hardware del dispositiu virtual, com per exemple la seva resolució de pantalla, la capacitat de la tarja SD, o la disponibilitat de GPS.
Amb aquestes passes, ja es tenen totes les eines necessàries per a començar a desenvolupar aplicacions per al sistema Android.

Una altra opció a tot el descrit en aquest punt de l'annexa, serà descarregar el *bundle* publicat en aquest mateix lloc web de l'SDK d'Android, el qual ja conté la darrera versió d'Eclipse i l'ADT integrats i degudament configurats, amb la qual cosa ens evitem realitzar la passa anterior (descàrrega i configuració d'Eclipse) i les posteriors (descàrrega, instal·lació i configuració de l'ADT i de les Plattforms Tools d'Android).